A.I Technical Report

Contents

[Introduction 3](#_Toc192317953)

[The Problem 4](#_Toc192317954)

[The Data 5](#_Toc192317955)

[Methodology 6](#_Toc192317956)

[Code Overview 7](#_Toc192317957)

[The Models 7](#_Toc192317958)

[Setup (Cleaning) 7](#_Toc192317959)

[Model Creation 9](#_Toc192317960)

[Rendering (Graphs) 12](#_Toc192317961)

[The Website 14](#_Toc192317962)

[Results/Findings 15](#_Toc192317963)

[The Data 15](#_Toc192317964)

[The Models 16](#_Toc192317965)

[Heatmaps 16](#_Toc192317966)

[Model Scores 17](#_Toc192317967)

[Conclusion 18](#_Toc192317968)

[References 19](#_Toc192317969)

# Introduction

To do.

# The Problem

When going through the process of conceptualising, writing, filming and producing a movie, it can be hard to say if the movie will do well. Merely throwing money at a movie’s production will not make it successful. For example, both “The 13th Warrior" and “Mortal Engines” were given budgets of around $150 million, but the former only grossed $62 million and the latter grossed a slightly better $84 million (Saha, 2023).

To assist this problem, this document will go over the creation of a simple machine learning model that will attempt to find a new movie’s potential success from the movie’s information (such as genre and age ratings). While a movie’s rating does not show the monetary success of a movie, it is hoped that this tool will help to show if a new movie would be popular or not to help possibly remove some of the ambiguity that comes with movie creation.

# The Data

The data for this project is sourced from the metadata of the online movie database website known as IMDb (Verma, n.d.). IMDb’s database contains the information of thousands of movies and allows users to see a movie’s rating, cast and the people who produced said movie. It also allows users to find other movies similar to ones they already like (IMDb, n.d.). But most importantly, IMDb gives each movie a rating based on user ratings rather than ratings from professional reviews. This helps to give a better idea on what movies are popular rather than just being considered “good”.

The data itself contains the following headers:

* name: The name of each movie.
* year: The year each movie was released.
* movie\_rated: The age rating of each movie.
* run\_length: Runtime of each movie.
* genres: The genres of each movie separated by a semicolon.
* release\_date: The date each movie was released.
* rating: The rating of each movie from IMDb.
* num\_raters: Number of people who rated each movie.
* num\_reviews: Number of people who reviewed each movie.

(Verma, n.d.)

# Methodology

For this project, the data will be loaded into the Python programming language and, through the use of the pandas library, will be loaded into tables. Once the data is loaded, it will be cleaned and normalised before then being used to create four models using K-Nearest Neighbours and four models using linear regression, with those four models being:

1. Age Rating
2. Runtime Length
3. Genre
4. Release Month

Each model will give its predicted user ratings. Then, the average of each of these user ratings will be finally given as the predicted user rating for the new movie. After this, graphs will be created using another Python library known as matplotlib in order to show how the models came to the predicted user rating. After the models have been created, a website will be created to allow the inputting of movie information and the prediction of the potential user rating.

# Code Overview

## The Models

### Setup (Cleaning)

A screen shot of a computer code

AI-generated content may be incorrect.To start, the data needs cleaning to ensure that all of the data is compatible and that none of the rows or columns are missing data. The data also needs to be normalised so that a computer can understand it. However, the data is already clean as there are no missing values or corruptions. Just in case, any rows with blanks are dropped using the Pandas negative operator to select all rows that do not have blanks [line 16] (Include Help, n.d.).

Alongside this, the “name”, “year”, “num\_raters” and “num\_reviews” columns are dropped from the dataset because these columns have little to no correlation with the user ratings.

(Continued…)



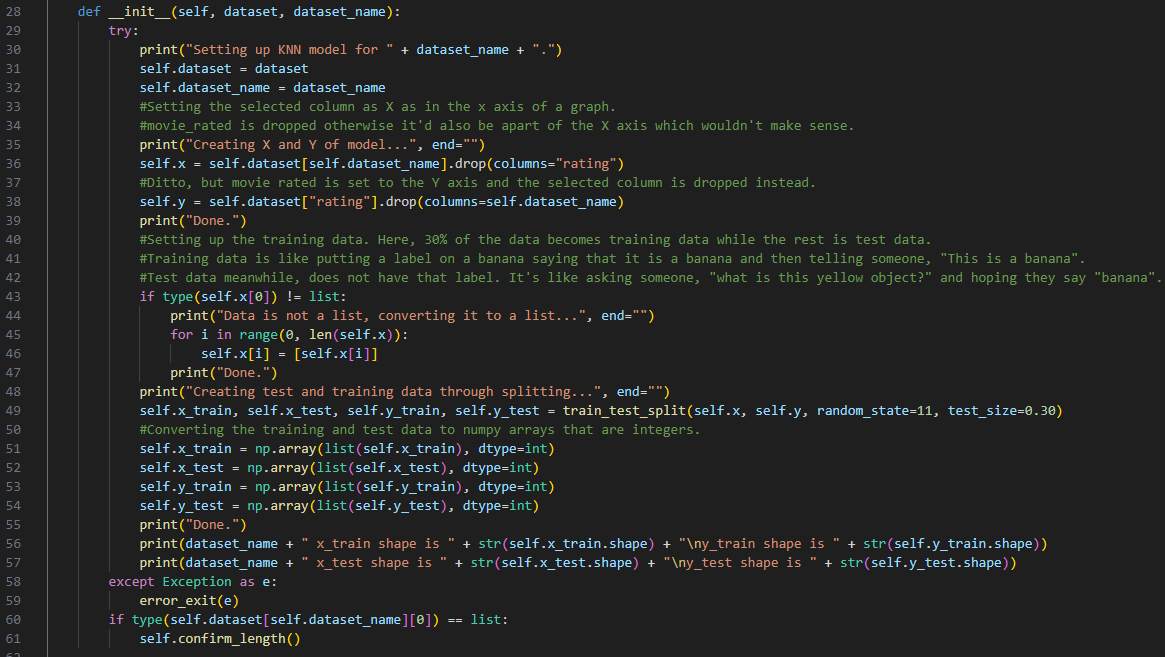
Next, the data needs to be normalised to ensure that the computer understands it.

For the genre and age rating columns, this means finding all unique values and representing them for each row as a list of Boolean values to represent the row containing that value. For example, for the genres column, there are 18 unique values, so the code creates a list of 18 Boolean values. However, the Linear Regression model specifically does not support this, so instead, all unique combinations of the genres and ages are put in a list and each row is given the index of that unique combination to use instead.

For the date column, only the month column is kept. The month is then converted to its numerical value (E.G: 6 for June). Lastly, for the runtime column, the text is dropped and the hours and minutes are converted into only minutes.

After this, the program them moves onto generating the models.

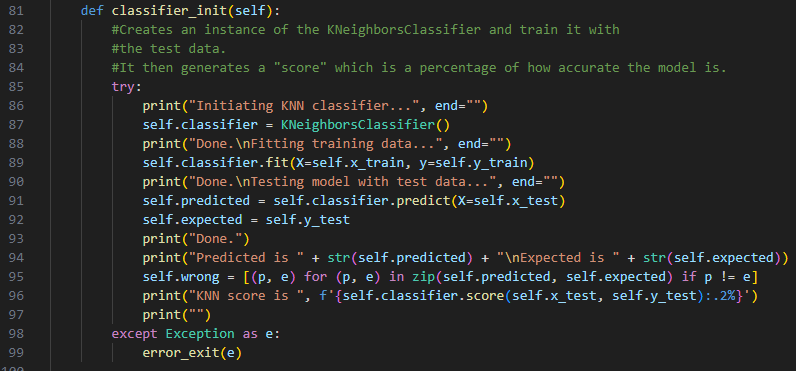
### Model Creation



For both the Linear Regression and KNN models, classes were used to make it simpler to execute commands and, more specifically, access data from the models. Both models start with an initialiser which sets an x value (belonging to the class) equal to the requested dataset column (E.G: The first one is genres) while dropping the “rating” column. Conversely, the y value is set to the genres column while dropping the other column. Without dropping the other column, the model might not work. After this, the program makes sure that the contents of each row are lists because the models only accept data as lists.

Lastly, the initialiser uses the scikit-learn function “train\_test\_split” which splits the data into training data and test data for the model. It takes the x and y values as inputs, but also random\_state (an integer that allows for splitting to be reproducible between machines) and test\_size (a float that determines the proportion of the data to use as test data) (Scikit Learn, n.d. -a). In the case of test size, it is set to 0.30 (30%) because any lower would reduce the validation of the model and any higher reduces the accuracy of the model.

(Continued…)



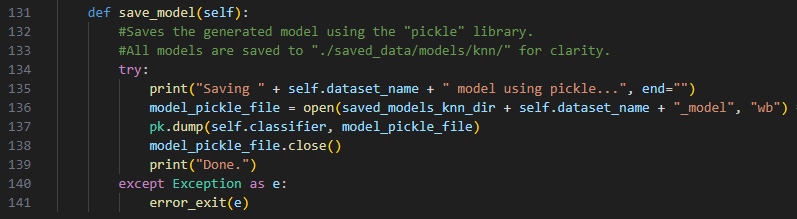
Next, the actual model is initialised whereupon it is fitted with the x training data and y training data created with train\_test\_split. The x values being what we want to train the model with and the y values being the target values (Scikit Learn, n.d. -b). Then, the model is tested using the predict method which takes the x test data and returns the nearest values learned from the training data (Scikit Learn, n.d. -b) Finally, we compare the results of the test (self.predicted) and the actual values (self.expected) to test how accurate the model is.

A screen shot of a computer code

AI-generated content may be incorrect.

With the model created and training having been done, a confusion matrix is created to help visualise the accuracy of the model by showing the model’s predictions compared to the actual values (Lu, n.d.). Finally, a classification report is also generated which details the precision, recall, f1-score and accuracy of the model.

(Continued…)



Lastly, the model is saved using the Python library “pickle” which saves the model to a binary file using serialisation (Python Docs, n.d.). This allows the model to be reloaded and used again on any machine by being de-serialised by the “pickle” library (Python Docs, n.d.).

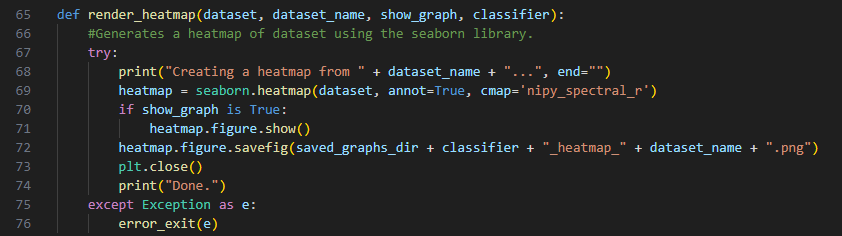
With that, the models are completed and can be deployed.

### Rendering (Graphs)

A screen shot of a computer code

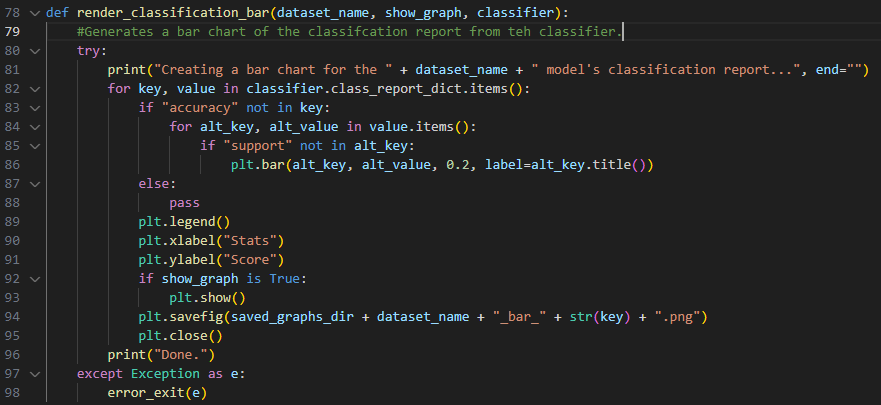
AI-generated content may be incorrect.

With the models having been created, the program finally moves on to rendering graphs from the datasets and the models. For the datasets, scatter graphs are used which are created with the above code. Scatter graphs are best suited to this because they can show all values on one graph clearly. This helps to estimate what the models might predict with a certain value. For this, the matplotlib library is used.



For the models themselves, heatmaps are generated to visualise the confusion matrixes in a clearer format. This uses the seaborn library and shows the options that the model is most likely to select (I.E: The options the model has the most support for) using colours to represent the likelihood of selection.

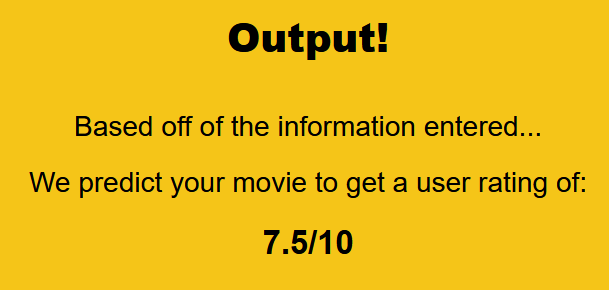
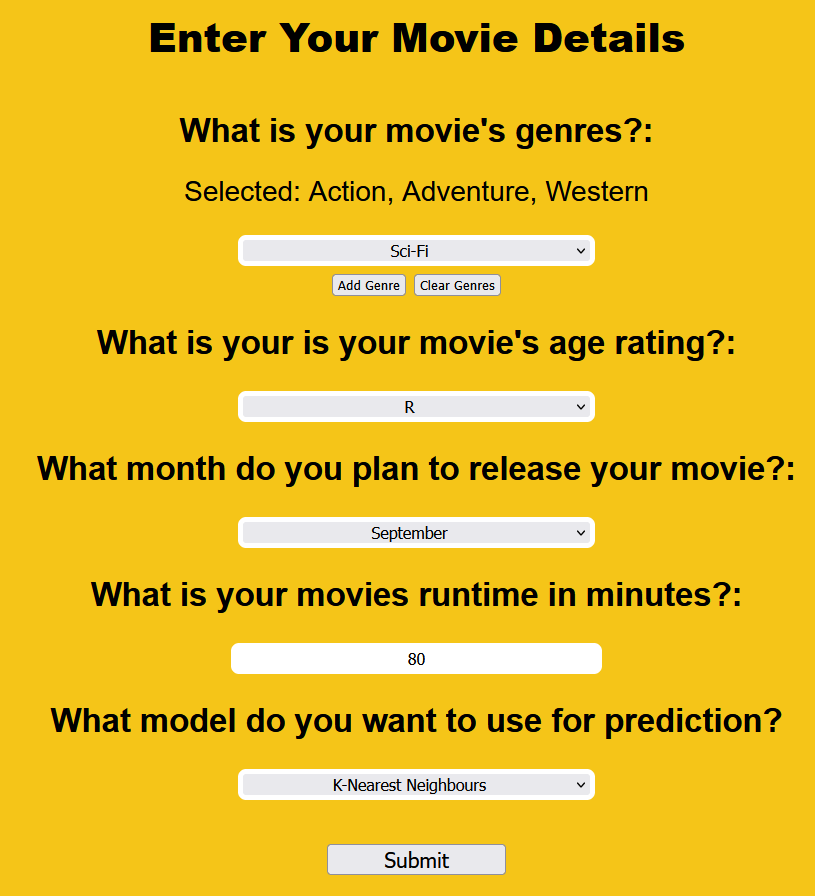
(Continued…)



Finally, the classification report of each model is rendered as a bar chart that shows the precision, recall and f1-scores of each model clearly.

## The Website

A screenshot of a movie details

AI-generated content may be incorrect. 

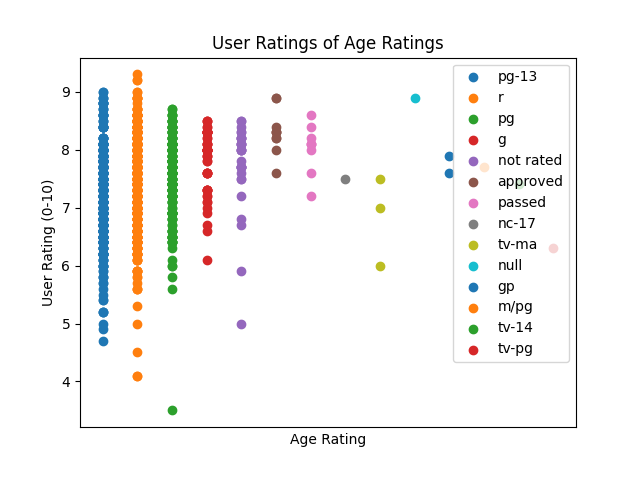
The website is a simple webapp made using the Flask library. It has three pages with one being the home page, the second being the page in which users can enter their movie’s information (and select either linear regression or KNN) and then a final page which shows the user rating the models predicted.

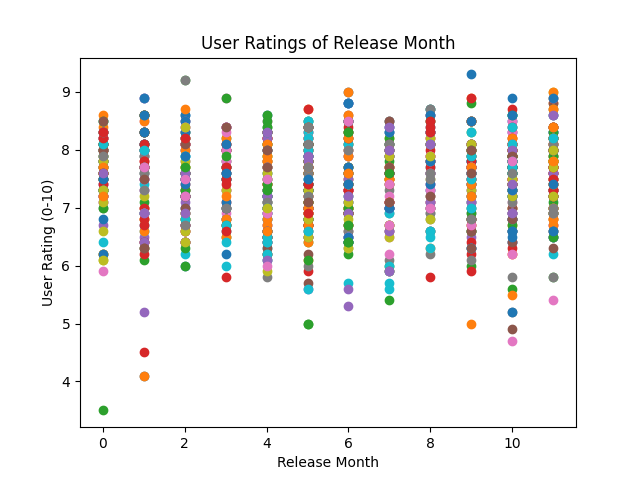
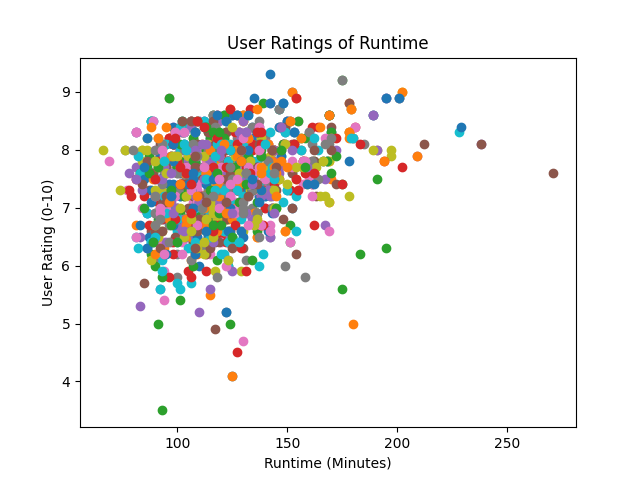
For this to work, the models are loaded from where they were saved and de-serialised by pickle (Python Docs, n.d.). Then, the data entered by the user is normalised and, using the predict method of the models, the user rating for each model is predicted. The average of these four predictions is then returned to the user as their movie’s predicted rating.

# Results/Findings

## The Data

A chart of dots with text

AI-generated content may be incorrect. 

After creating the models, the program creates four graphs of the data as it appears after having been normalised. To start, there is the genres graph. This graph shows all of the unique genres found by the program and what user rating they correspond with. Unfortunately, there is no clear pattern here apart from a few genres having oddly low ratings and a strange up and down “wave”.

Next, there is the age ratings graph. This shows that most of the data focusses on movies rated PG-13 and R of which these ratings cover a large range of user ratings. This does show that more data is needed for the other age ranges.

Penultimately, there is the release month graph. This shows that user ratings are mostly consistent for movies throughout the year. However, there is slight decrease in the middle of the year which would suggest that movies released near holidays are rated better, but not by a significant margin.

Lastly, there is the runtime graph. This graph very slightly suggests that, while movies before 150 minutes vary in rating, overall, movies get higher ratings as they get longer, before the ratings become lower again after 200 minutes. More data would be needed to prove this, however.

## The Models

### Heatmaps

### Model Scores

# Conclusion

# References

* IMDb. (n.d.). About IMDb. IMDb. <https://www.imdb.com/pressroom/about/?ref_=fea_eds_center-29_pr_related_1>
* Include Help. (n.d.). Python -Tilde Sign (~) in Pandas DataFrame. Include Help.
* Python Docs. (n.d.). Pickle – Python object serialization. Python Docs. <https://docs.python.org/3/library/pickle.html>
* Scikit Learn. (n.d. -a). train\_test\_split. Scikit Learn. <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html>
* Scikit Learn. (n.d. -b). KNeighborsClassifier. Scikit Learn. <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>
* Saha, P. (2023, November 10th). Hollywood: These Are The Most Expensive Flops In Movie History. CEOWORLD Magazine. <https://ceoworld.biz/2023/11/10/hollywood-these-are-the-most-expensive-flops-in-movie-history/>
* Verma, K. (n.d.). IMDb Movies Dataset. Kaggle. <https://www.kaggle.com/datasets/krishnanshverma/imdb-movies-dataset>
* Lu, X. (n.d.). Model Evaluation & Evaluation Metrics. Leeds Trinity University. <https://moodle.leedstrinity.net/pluginfile.php/745946/mod_folder/content/0/Week%207%20Model%20Evaluation%20%20Evaluation%20Metrics.pdf?forcedownload=1>